BitTorrent Final Project

Josh Suo, Veer Singh, Mosinmiloluwa Ojeyomi, Oluwatayo Odunlami, Andre Fonseca Bhandoola

Supported Features

- ***Core Features***
    - Communication with HTTP trackers
    - Download a file from official BitTorrent Clients

Design and Implementation

- **Language Choice:**
    - Initially we chose to start with the C language, as it is a powerful language for optimizing memory use and one we were all familiar with, but we realized how difficult implementing many of the trickier aspects of the information management would be in a language that requires such attention to the minutiae of bit management and the challenges that would then come with implementing TCP send and receive. Therefore, we switched to Python, as its ease of use and abstracting of certain inconvenient facts of memory management vastly increased our ability to focus on the logical steps required to implement the specs of the project.

- **Code Inputs:**
    - Our BitTorrent client takes as an initial input the port number it will be working with, and once it is running, it additionally takes the .torrent file it will be acting on as a command line argument. Once the program has its .torrent file, it begins connecting to the tracker and attempting to commence the download until the local peer possesses the entire file. However, as soon as the local peer possesses even a single piece of the request, it will make that piece available for other peers to download from as a seeder.
    - In a case where the wrong number of arguments are requested, the program will produce usage information

- **Program Outputs**
    - Program outputs are sent through STDOUT, and depending on the end result of the process, the program will output differently.
        - Upon successfully downloading a full .torrent file and joining the swarm, the program will output the torrent file details, list the number of peers contacted, and print out a happy success message.

        - If the .torrent file download encounters a problem, or the program at any point fails, the process will output the .torrent file details if possible, and will print out a sad failure message.

- **Libraries**
    - The only non-standard libraries we imported were the bencode and urllib.parse libraries to support basic functionality of decoding the .torrent files as well as interacting with the url functionality required to contact the tracker.

- **Program Structure**
    - Parsing:
    - Tracker Announcement:
    - Downloading:
    - Listening:
    - Uploading:

- **Challenges**
    - Our single biggest challenge was pulling the trigger on switching from C to Python. While Python is by far the easier language to work in, our group makeup was determined to stick it out in C for as long as possible, to the point that we were almost cutting it close by the time we decided to switch over from C.
    - We initially believed that we needed to implement many of the bencode libraries ourselves from scratch, a misstep that led to us wasting a decent deal of time on debugging pieces of code that we could have simply pulled fully functional from another library.

- **Known Bugs**

- **Group Member Contributions**
    - Parsing: Oluwatayo, Veer
    - HTTP GET requests: Mosinmiloluwa
    - TCP Functionality: Andre, Veer
    - Download: Oluwatayo, Mosinmiloluwa, Veer, Josh
    - Upload: Oluwatayo, Mosinmiloluwa, Josh, Andre